**Proximal Policy Optimization Algorithms**

<https://arxiv.org/pdf/1707.06347.pdf>

**0. INTRODUCTION**

|  |
| --- |
| Reinforcement learning에서 leading contender는 **deep Q learning, policy gradient method, trust region/natural policy gradient method**이다.   * 그러나 더 **scalable하고, data efficient하며 robust한 method**를 개발할 수 있는 여지가 있다.   + **Q learning**은 많은 간단한 문제들에서 실패하고, 부족하게 이해된다.   + **Vanilla policy gradient method**는 data efficiency가 낮고 robust하지 않다.   + **Trust region policy optimization (TRPO)**는 상대적으로 복잡하고, noise (dropout 등)를 포함하거나 parameter sharing을 포함하는 구조에서는 compatible하지 않다.   이 논문에서는,   * **여러 가지 서로 다른 버전의 surrogate (대체) objective를 비교**하고, * **clipped probability ratio를 사용한 버전의 성능이 가장 좋다**는 것을 보인다. * 또한, **PPO (proximal policy optimization)**를 몇 가지의 이전의 알고리즘과 비교한다. |

**1. BACKGROUND: Policy Optimization**

|  |
| --- |
| **<Policy Gradient Methods>**  Policy Gradient method는 **policy gradient의 estimator를 계산**한 다음 이것을 **stochastic gradient ascent 알고리즘에 적용**하는 것이다.   * 가장 많이 사용되는 gradient estimator는 다음과 같은 형태이다. * 여기서 **expectation** 는sampling과 optimization을 교대로 반복하는 알고리즘에서, **한정된 batch sample에 대한 empirical average**를 나타낸다. * 여기서 **estimator** **는 다음의 objective를 미분하여 얻어진다.**   여기서 같은 trajectory를 사용하여 를여러 단계의 optimization step을 적용하는 것이 매력적이지만, 실제로는 **well-justify되어 있지 않**고, **파괴적으로 많은 policy update**를 유발할 수 있다. |

**1. BACKGROUND: Policy Optimization (Cont.)**

|  |
| --- |
| **<Trust Region Methods>**  TRPO에서는 **objective function (surrogate objective)**이 다음과 같이 **policy update의 크기의 제약 조건에 따라 최대화**된다.  여기서 는 update 이전의 policy parameter의 벡터이다.  이 문제는 **objective에 대한 linear approximation과 constraint에 대한 quadratic approximation**을 생성한 후, **conjugate gradient 알고리즘**을 이용하여 approximate하게 해결할 수 있다.  TRPO를 justify하는 이론에서는 **constraint 대신 penalty를 사용할 것을 제안**한다. 예를 들어 다음의 unconstrained optimization problem을 해결하는 것이 될 수 있다.  TRPO는 penalty보다는 hard constraint를 이용하는데, 그 이유는 **여러 가지 서로 다른 문제에 대해서 성능이 좋은 의 값을 찾기 어렵기 때문**이다. |

**2. Clipped Surrogate Objective**

|  |
| --- |
| Probability ratio를 라고 하면, 여기서 이다. 여기서 **TRPO는 다음과 같은 surrogate objective를 최대화**한다.  여기서 **constraint 없이는 를 최대화하는 것이 과도하게 큰 policy update**로 이어질 수 있다. 따라서 이제 **objective를 어떻게 수정하고 policy change를 어떻게 penalize**할 것인지를 생각해 본다.  여기서 **main objective**는 다음과 같다. |

**2. Clipped Surrogate Objective (Cont.)**

|  |
| --- |
| 여기서 는 **probability ratio를 clip하여 surrogate objective를 수정**하고, 이때 **를 interval 의 바깥**으로 움직이는 것에 대한 **incentive를 제거**한다.   * 최종적으로 clipped, unclipped objective에서 최솟값을 이용하므로, **final objective는 unclipped objective에 대한 lower bound (pessimistic bound)**이다. * 이 scheme에서 **objective를 향상시킬 것으로 예상될 때만 probability ratio의 변화를 무시**할 수 있고, **objective를 감소시킬 때 include**한다. * Probability ratio r는 **advantage가 positive인지 negative인지**에 따라 또는 로 clip된다. 이때의 그래프는 다음과 같다.     아래 그림은 surrogate objective 에 대한 또 다른 source of intuition이다.   * **Continuous control problem**에서 **Policy update direction을 따라 interpolate**할 때, 몇 개의 object가 어떻게 달라지는지 보여준다. (proximal policy optimization) * **은 의 lower bound**이고, policy update에 대한 매우 큰 penalty를 가지고 있다는 것을 알 수 있다. |

**3. Adaptive KL Penalty Coefficient**

|  |
| --- |
| 다른 접근법은 **clipped surrogate objective에 대한 대안적 또는 추가적**인 접근 방법이다. 즉, **KL divergence에 대한 penalty**를 이용하고, **penalty coefficient**를 적용하여 **KL divergence 에 대한 target value**를 각 policy update마다 얻는 것이다. |

**3. Adaptive KL Penalty Coefficient (Cont.)**

|  |  |
| --- | --- |
| 이 논문에서는 **KL penalty가 clipped surrogate objective보다 더 좋지 않은 성능**을 보였다는 것을 발견했다. 그러나 그것이 중요한 baseline이기 때문에 그것을 이용했다.  여기서는 각 policy update에 대해 다음의 step을 적용했다.   |  | | --- | | **Minibatch SGD**를 몇 epoch 동안 사용하고, **KL-penalized objective**를 다음을 이용하여 최적화한다.  를 계산한다.   * 이면 를, 이면 를 적용한다.   이때 **update된 의 값은 다음 policy update에서 사용**된다. 여기서 **KL divergence가 와 크게 다를 때 policy update**를 종종 확인할 수 있다. 그러나 이 경우는 드물고, **대부분 가 빠르게 adjust된다.**   * Parameter 1.5와 2는 heuristic하게 결정된 것이지만 알고리즘은 이에 크게 민감하지 않다. | |

**4. Algorithm**

|  |
| --- |
| 이전 section에서의 surrogate loss는 **전형적인 policy gradient implementation으로부터 약간의 변동을 주어서 계산 및 미분**될 수 있다.   * Automatic differentation을 사용하는 구현을 위해서, 우리는  **대신 이나 을 loss로 사용**할 수 있고, 이 objective에 대해 **여러 step 동안 stochastic gradient ascent를 적용**할 수 있다. * Policy function과 value function 간에 **파라미터를 공유하는 신경망 구조**를 사용한다면, **policy surrogate와 value function error term을 결합**한 loss function을 사용해야 한다.   + 이때 다음과 같은 objective가 도출된다.   Policy gradient 구현의 한 가지 스타일(Mni-16)은 **recurrent neural network에 적합**한데, 다음과 같은 **advantage estimator를 필요**로 한다. |

**4. Algorithm (Cont.)**

|  |
| --- |
| **Fixed-length trajectory segment**를 사용하는 **proximal policy optimization (PPO)** 알고리즘은 다음과 같다.   * N개의 **parallel한 각 actor는 T timestep 동안 데이터를 수집**한다. * 그 다음에 이들 NT timestep 동안의 데이터에 대한 **surrogate loss를 구성 및 minibatch SGD를 이용하여 최적화**한다. |

**5. Experiments**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **<Comparison of Surrogate Objectives>**  먼저, 몇 개의 서로 다른 **surrogate objective들을 서로 다른 hyperparameter 상에서 테스트**한다.   * 여기서 **surrogate objective** 을 몇 개의 **natural variation과 ablated version**들과 비교한다.  |  |  | | --- | --- | | No clipping or penalty |  | | Clipping |  | | KL penalty (fixed or adaptive) | * Fixed penalty coefficient 또는 adaptive coefficient를 사용할 수 있다. |  * 여기서는 각 알고리즘에 따라 **다양하게 hyperparameter search**를 할 것이므로, **계산적으로 cheap한 benchmark**에서 알고리즘을 테스트한다.   + OpenAI Gym에 있는 7개의 simulated robotics task를 구현한다. * **Policy를 표현**하기 위해 **fully-connected MLP**를 사용한다.   + 이 MLP에는 64개의 unit이 있는 2개의 hidden layer와 tanh nonlinearity가 있으며, output은 Gaussian Distribution의 평균이다. * 각 알고리즘은 **7개의 모든 environment**에서 돌아가며, 이들 **각각에 3개의 random seed**가 있다. * 평가 방법은 **최근 100개의 episode에 대한 average total reward**를 이용하여 계산한 점수를 이용한다. |

**5. Experiments (Cont.)**

|  |
| --- |
| **Continuous control benchmark**에서 각 알고리즘의 **average normalized score**는 오른쪽 그림과 같다.  **<Comparison to Other Algorithms in the Continuous Domain>**  여기서는 **PPO method를 다른 몇 개의 method와 비교**할 것이다.   * Trust region policy optimization [Sch+15b] * Cross-entropy method (CEM) [SL06] * Vanilla policy gradient with adaptive stepsize, A2C [Mni+16] * A2C with trust region [Wan+16]   A2C, PPO에 대한 사항은 다음과 같다.   * **A2C는 advantage actor critic**을 의미하며, A3C의 synchronous version이고, 이것이 적어도 **asynchronous version보다는 성능이 좋다**는 것을 보여야 한다. * **PPO에서는 이전 section에서의 hyperparameter들을 이용**하며, 이때 이다.   여기서는 **PPO가 이전에 소개한 method들보다 거의 모든 continuous control environment에서 더 좋은 성능을 보인다**는 것을 증명한다. |

**5. Experiments (Cont.)**

|  |
| --- |
| **<Showcase in the Continuous Domain : Humanoid Running and Steering>**  High-dimensional continuous control problem에서의 PPO의 성능을 보이기 위하여, 3D humanoid를 포함한 problem의 집합에서 훈련시킨다.   * RoboschoolHumanoid (forward locomotion only) * RoboschoolHumanoidFlagrun (target의 위치가 200 timestep마다 랜덤하게 달라짐) * RoboschoolHumanoidFlagrunHarder       **<Comparison to Other Algorithms on the Atari Domain>**  **Arcade Learning Environment benchmark에서 PPO를 실행**하고 **A2C의 well-tuned implementation과 ACER와 비교**하였다.   * **모든 알고리즘에 대해서 서로 같은 policy network 구조**를 사용하였다. * PPO에서는 오른쪽의 표와 같이, PPO를 제외한 다른 알고리즘에 대해서는 성능을 최대화하도록 튜닝된 hyperparameter를 이용하였다. * **Average reward per episode over entire training period / over last 100 episode of training**을 고려하였다. |